Brief Announcement: Rapid Asynchronous Plurality Consensus

Robert Elsässer
University of Salzburg, Austria
elsa@cosy.sbg.ac.at

Tom Friedetzky
Durham University, U.K.
tom.friedetzky@dur.ac.uk

Dominik Kaaser
Universität Hamburg, Germany
dominik.kaaser@uni-hamburg.de

Frederik Mallmann-Trenn
École normale supérieure, France
Simon Fraser University, Canada
mallmann@di.ens.fr

Horst Trinker
University of Salzburg, Austria
horst.trinker@sbg.ac.at

ABSTRACT

We consider distributed plurality consensus on a complete graph of size n with k initial opinions in the following asynchronous communication model. Each node is equipped with a random Poisson clock with parameter $\lambda = 1$. Whenever a node's clock ticks, it samples some neighbors uniformly at random and adjusts its opinion according to the sample.

Distributed plurality consensus has been deeply studied in the synchronous communication model. A prominent example is the so-called Two-Choices protocol, where in each round, every node chooses two neighbors uniformly at random, and if the two sampled opinions coincide, then that opinion is adopted. This protocol is very efficient when $k = \mathrm{O}(n^\epsilon)$ for some small ε, we show that it converges to the initial plurality opinion within $\mathrm{O}(k \cdot \log n)$ rounds, w.h.p., as long as the initial difference between the largest and second largest opinion is $\Omega\left(\sqrt{n \log n}\right)$. On the negative side, we show that there are cases in which $\Omega(k)$ rounds are needed, w.h.p.

To beat this lower bound, we combine the Two-Choices protocol with push-pull broadcasting. We divide the process into several phases, where each phase consists of a two-choices round followed by several broadcasting rounds. Our main contribution is a non-trivial adaptation of this approach to the above asynchronous model. If the support of the most frequent opinion is at least $(1 + \varepsilon)$ times that of the second-most frequent one and $k = \mathrm{O}(\exp(\log n / \log\log n))$, then our protocol achieves the best possible run time of $\mathrm{O}(\log n)$, w.h.p. Key to our adaptation is that we relax full synchronicity by allowing $\mathrm{o}(n)$ nodes to be poorly synchronized, and the well synchronized nodes are only required to be within a certain time difference from one another. We enforce this "sufficient" synchronicity by introducing a novel gadget into the protocol. Other parts of the adaptation are made to work using arguments and techniques based on a Pólya urn model.

KEYWORDS

Plurality Consensus; Distributed Randomized Algorithms; Asynchronicity

PODC '17, July 25-27, 2017, Washington, DC, USA

ACM ISBN 978-1-4503-4992-5/17/07.
https://doi.org/10.1145/3087801.3087860

1 INTRODUCTION

We consider the following plurality consensus process on the clique K_n of size n. Initially, the nodes are partitioned into k groups representing k colors $C_1, \ldots, C_k$. We will denote the number of nodes having color C_j as c_j. W.l.o.g., we assume that colors are ordered in descending order such that $c_1 \geq c_2 \geq \cdots \geq c_k$.

Synchronous Model. In the synchronous model we assume that the protocol operates in discrete rounds. In each round, the nodes may sample other nodes uniformly at random and then simultaneously change their opinion as a function of the observed samples. One prominent example here is the Two-Choices process [2] where in each round every node samples two nodes chosen uniformly at random, with replacement. If the chosen nodes' colors coincide, then the node adopts this color.

Asynchronous Model. In the continuous asynchronous model, every node is equipped with a random clock which ticks according to a Poisson distribution with parameter $\lambda = 1$. Whenever a node ticks, it samples nodes uniformly at random and updates its opinion based on the sampled values.

Instead of considering the asynchronous parallel process in continuous time, we rather analyze the process in the sequential model. In this sequential model, we assume that a discrete time is given by the sequence of ticks, and at any of the discrete time steps, a node is selected uniformly at random from the set of all nodes to perform its task. The two models lead to the same run time, see [4].

1.1 Our Contributions

We consider a modification of the Two-Choices protocol to design an efficient distributed voting algorithm, allowing a large number of different opinions in the asynchronous settings. So far, most work in this area concentrated on the synchronous communication model. As we see below, the Two-Choices protocol has certain limitations – even in this synchronous setting.

Limits of the Two-Choices Approach. The Two-Choices protocol seems to be very efficient if the number of colors is two [2]. The following result can be seen as an extension of Cooper et al. [2] on the complete graph to more than two opinions.

Theorem 1.1. *Consider the synchronous model. Let $G = K_n$ be the complete graph with n nodes. Let $k = \mathrm{O}(n^\epsilon)$ be the number of opinions for some small constant $\varepsilon > 0$. The Two-Choices plurality consensus process converges with high probability*[1] *to C_1 within $\mathrm{O}(n/c_1 \cdot \log n)$*

[1] The expression *with high probability* means a probability of at least $1 - n^{-\Omega(1)}$.

rounds, if the initial bias is at least $c_1 - c_2 \geq z \cdot \sqrt{n \log n}$ *for some constant* z*. Furthermore, if we assume that* $c_1 - c_2 = z \cdot \sqrt{n \log n}$ *for some constant* z *and* $c_2 = \cdots = c_k$*, then the Two-Choices protocol requires* $\Omega(n/c_1 + \log n)$ *rounds in expectation to converge.*

Interestingly, the required initial gap does not depend on the number of opinions present. Moreover, if $c_1 - c_2 = \mathrm{O}(\sqrt{n})$, then C_2 wins with constant probability. To overcome the $\Omega(k)$ lower bound in general, we combine the Two-Choices process with a rumor spreading algorithm. We first consider this approach in the synchronous setting and denote the corresponding algorithm by `OneExtraBit`. For this, we investigate a slightly modified model called the *memory model*. In this model, we allow each node to transmit one additional bit. This allows us to reduce the run time from $\mathrm{O}(n/c_1 \cdot \log n)$ to $\mathrm{O}((\log(c_1/(c_1 - c_2)) + \log\log n) \cdot (\log k + \log\log n)) = \mathrm{O}\left(\log^2 n\right)$, and the dominating color still wins with high probability, while the initial bias needs only to be slightly larger.

Theorem 1.2. *Consider the synchronous model. Let* $G = K_n$ *be the complete graph with* n *nodes. Let* $k = \mathrm{O}(n^{\varepsilon})$ *be the number of opinions for some small constant* $\varepsilon > 0$*. Assume* $c_1 - c_2 \geq z \cdot \sqrt{n} \log^{3/2} n$ *for some constant* z*, then the plurality consensus process* `OneExtraBit` *on* G *converges within*

$$\mathrm{O}((\log(c_1/(c_1 - c_2)) + \log\log n) \cdot (\log k + \log\log n))$$

rounds to C_1*, with high probability.*

Coming from a different angle, essentially the same result was obtained independently by Berenbrink et al. [1] and by Ghaffari and Parter [3]. To obtain our main result, we will generalize this approach to the asynchronous communication model.

Our main contribution is an adaptation of the algorithm `OneExtraBit` to the asynchronous setting. In the sequential asynchronous model, many nodes may remain *unselected* for up to $\mathrm{O}(\log n)$ time, which implies that no algorithm can converge in $\mathrm{o}(\log n)$ time. Thus, our aim is to construct a protocol that solves plurality consensus in $\mathrm{O}(\log n)$ time. We show that if the difference between the numbers of the largest two opinions is at least $\Omega(c_2)$, where c_2 is the size of the second largest opinion, and $k = n^{\mathrm{O}(1/\log\log n)}$, then our algorithm solves plurality consensus and achieves the best possible run time of $\mathrm{O}(\log n)$, provided a node is allowed to communicate with at most constantly many other nodes in a step. Our result is formally stated in the following theorem.

Theorem 1.3. *Consider the asynchronous model. Let* $G = K_n$ *be the complete graph with* n *nodes. Let* $k = \mathrm{O}(\exp(\log n/\log\log n))$ *be the number of opinions. Let* $\varepsilon > 0$ *be a constant. Assume* $c_1 \geq (1 + \varepsilon) \cdot c_i$ *for all* $i \geq 2$*, then the asynchronous plurality consensus process on* G *converges within time* $\Theta(\log n)$ *to the majority opinion* C_1*, with high probability.*

2 SYNCHRONOUS CONSENSUS

In order to achieve a polylogarithmic run time we combine the two-choices process with the speed of broadcasting. More specifically, the protocol consists of $\Theta(\log(n/c_1) + \log\log n)$ phases which in turn consist of two sub-phases: (i) one round of the Two-Choices process and (ii) several rounds of the so-called Bit-Propagation sub-phase in which each node that changed its opinion during the preceding two-choice step broadcasts its new opinion.

More precisely, we equip each node with an additional bit of memory which is set to True if and only if the node changed its opinion in the Two-Choices sub-phase. In the Bit-Propagation sub-phase, each node u samples nodes randomly until a node v with its bit set to True is found. Then u adopts v's opinion and sets its own bit to True, which means that subsequently any node sampling u will set their bit as well.

The first sub-phase ensures that after the Two-Choices round the number of nodes holding opinion C_1 and having their bit set to True is concentrated around c_1^2/n. After the Bit-Propagation sub-phase all nodes will have their bit set, and the size of C_j's support is concentrated around c_j^2/x, where x is the total number of bits set after the Two-Choices sub-phase. This is enough to show that at the end of the phase after $\mathrm{O}(\log(n/c_1))$ rounds the difference between C_1 and any opinion $C_j \neq C_1$ increases quadratically such that $c_1'/c_j' \geq (1 - \mathrm{o}(1))c_1^2/c_j^2$. Due to the quadratic growth in the difference between C_1 and every other opinion, the number of required phases is only of order $\Theta(\log(n/c_1) + \log\log n)$. We assume that every node is aware of (upper bounds on) n and k, allowing us to use these values within the algorithm, and in particular to run it in multiple phases of length $\Theta(\log k + \log\log n)$ each.

3 TOWARDS ASYNCHRONOUS CONSENSUS

Recall that the key to the speed of the synchronous algorithm is the combination of the two-choice process with an information dissemination process. However, this interweaving of these processes requires that the nodes execute the sub-phases simultaneously. While this is trivially the case in the synchronous setting, it is extremely unlikely in the asynchronous setting, since the numbers of ticks of different nodes may differ by up to $\mathrm{O}(\log n)$.

To overcome this restriction, we adopt the following weaker notion of synchronicity. At any time we only require a $(1 - \mathrm{o}(1))$ fraction of the nodes to be *almost synchronous*. This relaxes full synchronicity in three ways: First, nodes are only almost synchronous, meaning that for any two nodes their working times may differ by up to $\Delta = \Theta(\log n/\log\log n)$. Secondly, we allow $\mathrm{o}(n)$ nodes to be poorly synchronized. Finally, we require this to hold only with high probability.

The above notion does not require the nodes to synchronize actively per se, since their number of ticks is to some extent concentrated even without active synchronization. However, it turns out that without synchronizing perpetually, the number of poorly synchronized nodes in each phase will become larger than the initial bias towards the plurality opinion $c_1 - c_2$ and could therefore influence the consensus significantly. We thus actively synchronize nodes at the end of each phase to decrease the fraction of poorly synchronized nodes such that their number is in $\mathrm{o}(c_1 - c_2)$, resulting in a negligible influence of those nodes.

Once several technical challenges are resolved, the resulting weak synchronicity allows us to reuse the high-level structure of the synchronous algorithm. As in the synchronous case, the asynchronous protocol consists of one Two-Choices sub-phase and one Bit-Propagation sub-phase, the latter of which propagates the

choices of the Two-Choices phase to all nodes in the network. In addition to these sub-phases we have a third sub-phase in which we synchronize nodes.

After executing the first two sub-phases, the relative difference between C_1 and any opinion $C_j \neq C_1$ increases quadratically and thus we only require $O(\log \log n)$ such phases. Each of the sub-phases has a length of $O(\log n / \log \log n)$, amounting to a total runtime of $O(\log n)$. While the asynchronous version may look very similar to the synchronous protocol, the analysis differs significantly from the synchronous case, in both approach and technical execution.

3.1 The Asynchronous Protocol

Our asynchronous protocol consists of two parts. The goal of the first part is to increase the number of nodes of color C_1 to at least $c_1 \geq (1-\varepsilon) \cdot n$ for some small constant ε. Once the execution of the first part has finished, the nodes execute a simple two-choices algorithm in an asynchronous manner, after which C_1 wins with high probability.

The algorithm operates in multiple phases. Each of these phases is split into three sub-phases. Each sub-phase consists of multiple blocks of length Δ each. During these sub-phases there are multiple blocks of instructions where nodes literally *do nothing*. These do-nothing-blocks are used, in combination with the following result on synchronicity, to ensure that a large fraction of nodes executes critical instructions at almost the same time. That is, for a large fraction of nodes we will show that these nodes execute instructions as if they were bulk synchronized, which they clearly are not.

The first phase is the Two-Choices sub-phase, which consists of two instructions, the Two-Choices step and the commit step. In the Two-Choices step, every node samples two neighbors uniformly at random. If and only if these neighbors' colors coincide, the node sets an intermediate color to the neighbors' colors. In the commit step, nodes change their color if they have their intermediate color set and then set their bit accordingly. The second phase is the Bit-Propagation sub-phase in which all nodes sample $O(\log n / \log \log n)$ times another node: As soon as node u samples a node v with a bit set, u also sets its bit and adopts v's color. At the beginning of the Bit-Propagation sub-phase, only a small fraction of the nodes will have their bit set. Their number grows to almost all nodes at the end. The crucial property is that among the nodes which have their bit set, a very large fraction supports C_1, while in the first round the number of nodes having their bit set may be as small as n/k in expectation. By modeling the process as a Pólya urn process and by using martingale techniques, we show that the distribution of colors among the nodes which set a bit after the Two-Choices sub-phase remains almost unchanged at the end of the Bit-Propagation sub-phase – the purpose of the Bit-Propagation sub-phase is to grow the fraction of nodes supporting C_1. Finally, in the third phase, all nodes execute the so-called *Sync Gadget*. In this gadget, nodes adjust their *working time* in order to synchronize.

Weak Perpetual Synchronization. In the asynchronous algorithm, when a node is selected to tick, all operations are performed based on the node's current working time. In contrast, the real time of a node is used to always count the total number of ticks performed so far by this node. The Sync Gadget consists of a sampling sub-phase of length $\log^3 \log n$. During these ticks, every node samples a neighbor uniformly at random and collects the real time T_u of the sampled neighbor u. Additionally, the node increments all real times sampled so far by 1 until a so-called jump step is executed. At the end of the phase at the jump step, after a proper waiting time, the node sets its working time to the median of the samples.

3.2 The Endgame

At the end of the first part, at least $(1-\varepsilon) \cdot n$ nodes have color C_1. We apply martingale techniques and drift theory to show that all nodes will have adopted C_1 before the first node finishes the execution of the second part of the algorithm, with high probability.

4 DISCUSSION

Our algorithm solves plurality consensus in the asynchronous setting in the best possible asymptotic run time in the setting where the number of opinions k is bounded by $\exp(\log n / \log \log n)$. It remains an open question whether there exists an algorithm with the same run time allowing for $k = O(n^{\varepsilon})$ opinions; we note that even in the synchronous setting this questions is open.

We believe that the concept of weak synchronicity (including the Sync Gadget and the tactical waiting) as well as our analysis techniques may well prove to be of independent interest.

We showed our main result assuming independent Poisson clocks with parameter 1. However, our techniques should carry over to a much more general setting as well. Moreover, we assumed that once a node contacts another node, it receives that node's response without any delay. This assumption, however, might be unrealistic in real networks (or other models of asynchronicity). We may address this issue by extending our model to allow for response delays following some exponential distribution with constant parameter (which need not be 1, but must be independent of n).

Finally, we feel that the ideas presented here may be applicable to the adaptation of synchronous protocols to asynchronous settings for a much wider class of problems, perhaps even eventually leading to a generic framework.

REFERENCES

[1] Petra Berenbrink, Tom Friedetzky, George Giakkoupis, and Peter Kling. 2016. Efficient Plurality Consensus, or: The benefits of cleaning up from time to time. In *Proc. ICALP '16.* 136:1–136:14.

[2] Colin Cooper, Robert Elsässer, and Tomasz Radzik. 2014. The Power of Two Choices in Distributed Voting. In *Proc. ICALP '14.* 435–446.

[3] Mohsen Ghaffari and Merav Parter. 2016. A Polylogarithmic Gossip Algorithm for Plurality Consensus. In *Proc. PODC '16.* 117–126.

[4] Damon Mosk-Aoyama and Devavrat Shah. 2008. Fast Distributed Algorithms for Computing Seperable Functions. *IEEE Transactions on Information Theory* 54, 7 (2008), 2997–3007.